

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 2 年 1 2 月 1 8 日
Date of Application:

出 願 番 号 特 願 2 0 0 2 - 3 6 6 3 3 7
Application Number:
[ST. 10/C]: [J P 2 0 0 2 - 3 6 6 3 3 7]

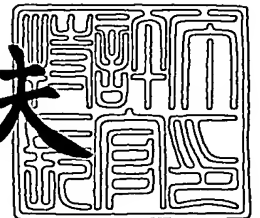
出 願 人 株 式 会 社 イ ン テ ッ ク ・ ネ ッ ト コ ア
Applicant(s):



2 0 0 3 年 1 2 月 5 日

特 許 庁 長 官
Commissioner,
Japan Patent Office

今 井 康 夫



出 証 番 号 出 証 特 2 0 0 3 - 3 1 0 1 1 1 7



【書類名】 特許願

【整理番号】 INET-001

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 19/00

【発明者】

 【住所又は居所】 富山県 高岡市 泉が丘 3 3 0 0 - 7 4

 【氏名】 中川 郁夫

【発明者】

 【住所又は居所】 千葉県 市川市 柏井町 1 - 1 7 9 9

 【氏名】 永見 健一

【特許出願人】

 【住所又は居所】 東京都 江東区 新砂 一丁目 3 番 3 号

 【氏名又は名称】 株式会社 インテック・ネットコア

【代理人】

 【識別番号】 100109553

 【弁理士】

 【氏名又は名称】 工藤 一郎

【手数料の表示】

 【予納台帳番号】 100322

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

 【物件名】 図面 1

 【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 計算装置、計算プログラム及び計算方法

【特許請求の範囲】

【請求項 1】 計算の実行のための実行部を指示するためのポインタを保持するポインタ保持部と、

ポインタ保持部に保持されるポインタを所定の変更条件に従って変更するポインタ管理部と、

前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能なデータ保持部と、

所定の実行条件に従って、前記データ保持部で保持されているデータを利用して、前記ポインタ保持部に保持されているポインタで指示される実行部により計算の実行をさせるための実行命令部と、

を有する計算装置。

【請求項 2】 前記ポインタ管理部は、所定の条件に従ってポインタの追加と削除とを行なう機能をさらに有する請求項 1 に記載の計算装置。

【請求項 3】 前記ポインタ管理部は、実行部の再読込をする実行部再読込手段を有し、

前記実行部再読込手段により再読込された実行部を指示するように前記ポインタを変更することを特徴とする請求項 1 に記載の計算装置。

【請求項 4】 前記実行部再読込手段は、データ保持部に保持されているデータを変換するためのデータ変換実行部をも読み込みことを特徴とする請求項 3 に記載の計算装置。

【請求項 5】 前記データはバージョン情報を保持し、前記実行部は、前記バージョン情報に応じた計算の実行を行なうことを特徴とする請求項 3 に記載の計算装置。

【請求項 6】 前記ポインタ管理部は、実行部を追加する実行部追加手段を有し、

前記実行部追加手段により追加された実行部を指示するポインタを前記ポインタ保持部に追加することを特徴とする請求項 2 に記載の計算装置。

【請求項 7】 ポインタ管理部は、実行部を削除する実行部削除手段を有し、前記実行部追加手段により削除された実行部を指示している前記ポインタの削除又は変更を行なうことを特徴とする請求項 2 に記載の計算装置。

【請求項 8】 計算の実行のための実行部を指示するためのポインタを保持するポインタ保持ステップと、

ポインタ保持ステップにより保持されたポインタを所定の変更条件に従って変更するポインタ管理ステップと、

前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能に保持するデータ保持ステップと、

所定の実行条件に従ってデータ保持ステップで保持されたデータを利用して、前記ポインタ保持ステップにより保持されたポインタで指示される実行部により計算の実行をさせるための実行命令ステップと、

を計算機に実行させるための計算プログラム。

【請求項 9】 計算の実行のための実行部を指示するためのポインタを保持するポインタ保持ステップと、

ポインタ保持ステップにより保持されたポインタを所定の変更条件に従って変更するポインタ管理ステップと、

前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能に保持するデータ保持ステップと、

所定の実行条件に従ってデータ保持ステップで保持されたデータを利用して、前記ポインタ保持ステップにより保持されたポインタで指示される実行部により計算の実行をさせるための実行命令ステップと、

を含む計算方法。

【発明の詳細な説明】

【 0 0 0 1 】

【発明の属する技術分野】

本発明は、計算装置、計算プログラム及び計算方法に関する。特に、ポインタにより計算を実行する実行部を指示し、その特定された実行部により、計算を実行するものに関する。

【0002】

【従来の技術】

従来、計算機の中でプログラムを実行する場合は、定義されていない関数（function）、手続き（procedure）、変数（variable）があつてはならず、例えば、高級プログラミング言語で記述されたプログラムをコンパイルして実行可能なオブジェクトを得るには、プログラムで参照される全ての関数、手続き、変数が定義されている必要があつた。

【0003】

ここに「プログラムで参照される全ての関数、手続き、変数が定義されている必要がある」とは、実行可能なオブジェクトから参照される関数、手続き、がコード（インストラクション）部分に存在し、また、実行可能なオブジェクトから参照される変数が保持する値がデータ部分に存在することを意味する。「コード（インストラクション）」、「データ」については、以下の段落で説明する。

【0004】

図23は、プログラムを計算機で実行する際に使用されるアドレス空間2300を模式的に示している。アドレス空間2300の上方が小さいアドレスに対応し、下方が大きいアドレスに対応するとすると、アドレスが小さい部分には、コード（インストラクション）とデータが保持される。その次にヒープと呼ばれる領域がある。また、アドレスが大きい方からスタックの領域が伸びている。

【0005】

コード（インストラクション）は、関数や手続きを実行するための計算機への命令を含む部分であり、コードの部分に含まれる命令で参照される関数、手続きがアドレス空間内に配置されるようになっていなければならない。またデータは、プログラミング言語で定義される大域変数の値を保持する。ヒープは、プログラムの実行に伴い、動的に確保されるメモリの領域であり、メモリがmalloc関数などにより動的に確保されると、矢印2301が示すように、ヒープの一端がアドレスの大きい方に向かって伸びて行く。スタックは、関数や手続きを呼び出すときに記憶すべき戻りアドレスを蓄積したり、局所変数を保持したりするためのメモリ領域である。スタックは、関数や手続きを呼び出すと、矢印230

2 が示すように、アドレスの小さい方に向かって伸びて行く。

【0006】

しかし、近年、共有ライブラリの技術が使用可能となり、実行可能なオブジェクトを得る場合には、共有ライブラリの中に定義されている関数、手続き、変数については、コンパイル時に定義されている必要がなくなった（例えば、特許文献1 参照。）。これにより、実行可能なオブジェクトが実際に実行されるときに、未定義の関数、手続き、変数が定義を含む共有ライブラリが存在すればよくなり、実行可能なオブジェクトに含まれる未定義の関数などは、実行開始時に共有ライブラリと動的にリンクが行なわれることによりアドレスが解決されるようになった。

【0007】

共有ライブラリの採用により、例えば、ライブラリ中の関数のバグの修正を反映するための再コンパイルが不要となったり、実行可能オブジェクトのサイズが小さくなったりするという効果が得られる。

【0008】

なお、共有ライブラリにより提供される関数、手続きを実行するための命令や、共有ライブラリにより提供される変数の値は、ヒープとスタックの間のメモリ空間に配置されるようになる。特に、そのメモリ空間は、他のプロセスと共有される共有メモリとすることが一般的である。

【0009】

【特許文献1】

特開平06-332675号公報（第2-3 ページ）

【発明が解決しようとする課題】

しかしながら、共有ライブラリを使用したとしても、実行可能なオブジェクトを実行している途中で、関数や手続きの定義を入れ換えることはできない。このため、例えば、バグが発見された場合、実行可能なオブジェクトの実行を停止させ、新たな実行可能なオブジェクト、あるいは、共有ライブラリを入れ換えて、プログラムを再起動する必要がある。

【0010】

このことは、計算機によるサービスが必要不可欠となった現代において、問題をもたらす。すなわち、実行可能なオブジェクトや共有ライブラリの入れ換えのために、そのオブジェクトや共有ライブラリにより提供されているサービスの提供を中断しなければならない。

【0011】

このようなサービスの中断は、例えば、ウェブサーバで24時間絶え間なくサービスを提供する場合に問題となる。

【0012】

また、このようなサービスの中断は、特に、インターネットなどの通信に用いられているルータにおいて、特に問題となる。

【0013】

図22は、ルータ2203、2204、2205、2206によりsite-Aとsite-Bとが通信可能な状態を例示している。この場合、実線の矢印のようにsite-Bからsite-Aへデータが送信されるようにするためには、site-Aへ至る経路情報を各ルータ2203、2204、2205、2206が持っている必要がある。このため、破線で示した向きに経路情報が流れる。また、同様にsite-Aからsite-Bへデータが送信できるようにするためには、実線で示した向きに経路情報が流れる必要がある。すなわち、ルータは、隣り合うルータと経路情報を交換し合うことになる。

【0014】

このように通信に用いられるルータで動作する実行可能なオブジェクト（「ファームウェア」と呼ばれる場合がある。）を入れ換えるために、ルータを停止させると、その間、そのルータを介する通信が途絶してしまう。また、上述したように、ルータ間では、経路情報の交換が行なわれ、この交換により、ルータは、特定のサイトへデータを送るための最短の経路などの情報を計算し、その結果を蓄積しているが、ルータを停止させると、蓄積した情報が消去してしまう。このため、再起動してから再び情報を計算・蓄積しなければならなくなり、停止前の状態に復旧するのに時間が必要となる。

【0015】

ルータによる最短経路の情報の計算について、現在用いられているアルゴリズムでは、ルータの数をN、通信路の数をLとすると、以下の式のオーダーの計算量が必要である。

【0016】

【式1】

$$N \times (N + L)$$

特に、Nの二乗に比例したオーダーとなるため、ルータの広く普及している現在では、復旧のためにはかなりの計算量が必要となり、それだけ復旧に時間がかかることになる。

【0017】

また、このルータのファームウェアのバージョンアップの問題は、特にインターネットサービスプロバイダにおいて、顕著となる。一つのインターネットサービスプロバイダが、数千の台数のルータを有しているのは珍しくない。例えば、あるインターネットサービスプロバイダに五千台のルータがあり、ルータのファームウェアのバージョンアップが年三回あるとすると、一年で延べ一万五千回のバージョンアップを行なう必要がある。平均すると一日約五十台のルータの停止を行ない、バージョンアップを行なうことになり、ルータの停止の問題は深刻である。

【0018】

そこで、本発明では、プログラムの実行を停止させることなく、関数や手続きなどを入れ換えることができるようにすることを目的とする。

【0019】

【課題を解決するための手段】

この課題を解決するために、本発明は、第一に、計算の実行のための実行部を指示するポインタを保持し、そのポインタを所定の条件に従って変更し、その変更によっても実行に利用されるデータをそのまま保持可能であり、また、所定の実行条件に従って、保持されているデータを利用して、ポインタで指示される実行部により計算の実行をさせる計算装置を提供する。

【0020】

これにより、ポインタを変更することにより、実行がされる実行部を変更することができる。また、ポインタが変更されても、データはそのまま保持されているので、継続的にサービスを提供することができる。

【0021】

第二に、所定の条件に従って、ポインタの追加と削除を行なう計算装置を提供する。

【0022】

これにより、新しいポインタを追加することにより、今まで実行がされなかった実行部を実行することができたり、既に存在する実行部が実行されないようにしたりすることができる。

【0023】

第三に、実行部の再読込を行ない、再読込がされた実行部をポインタが指示するようにする計算装置を提供する。

【0024】

これにより、例えば、バグなどを有する実行部の代わりに、バグが除去された実行部が実行されるようにすることができる。

【0025】

第四に、実行部を追加し、追加された実行部をポインタが指示するようにする計算装置を提供する。

【0026】

これにより、サービスの提供を停止することなく、新しい機能を追加することができる。

【0027】

第五に、実行部を削除し、削除された実行部を指示しているポインタの削除または変更を行なう計算装置を提供する。

【0028】

これにより、不要となったり、バグがあったりなどして使用できない実行部を削除することができる。

【0029】

また、本発明では、このような計算装置を計算機により実現するための計算プログラムも提供される。また、このような計算装置や計算プログラムなどにより用いられる計算方法も提供される。

【0030】

【発明の実施の形態】

以下、本発明の実施の形態について、図を用いて説明する。なお、本発明は、これら実施の形態に何ら限定されるものではなく、その要旨を逸脱しない範囲において、種々なる態様で実施し得る。

【0031】

(実施形態1 (主に請求項1、8、9に関連する))

本発明の実施形態1においては、計算の実行のための実行部を指示するポインタを保持し、そのポインタを所定の条件に従って変更し、その変更によっても実行に利用されるデータをそのまま保持可能であり、また、所定の実行条件に従って、保持されているデータを利用して、ポインタで指示される実行部により計算の実行をさせる計算装置、計算プログラム及び計算方法が提供される。

【0032】

(実施形態1：構成)

図1は、実施形態1における計算装置の機能ブロック図を例示している。計算装置100は、ポインタ保持部101と、ポインタ管理部102と、データ保持部103と、実行司令部104と、を有する。

【0033】

(実施形態1：ポインタ保持部)

「ポインタ保持部」101は、実行部を指示するためのポインタを保持する。「実行部」とは、計算の実行のための部である。

【0034】

図2は、ポインタ保持部と実行部との関係を例示している。ポインタ保持部101の中にポインタ201が存在し、実行部202を指示している。

【0035】

(実施形態1：ポインタ保持部：アドレス空間での例)

図 3 は、ポインタ保持部と実行部との関係がアドレス空間上でどのようなになっているかを例示する。アドレス空間 3 0 0 の中に、実行部に対応する部分として斜線を引いた部分 3 0 1 が存在する。この部分 3 0 1 が実行部に該当し、例えば、関数を実行するための命令が格納される。符号 3 0 2 を付けた部分がポインタ保持部に該当する。この部分には、例えば、斜線を引いた部分 3 0 1 の最初のアドレスが格納される。

【 0 0 3 6 】

(実施形態 1：ポインタ保持部：プログラミング言語での表現)

図 4 は、プログラミング言語上で、ポインタ保持部と実行部とがどのように表現されているかを例示する。図 4 では、プログラミング言語としては、C 言語を用いている。

【 0 0 3 7 】

(a) の部分が、変数 f_p を、ポインタを保持する大域変数であるポインタ変数として定義している。この変数 f_p がポインタ保持部となる。

【 0 0 3 8 】

(b) の部分が関数 f の定義である。この関数が実行部に相当する。

【 0 0 3 9 】

(c) の部分は関数 $main$ の定義である。C 言語では、 $main$ という名前の関数は特別な意味を持ち、プログラムの実行の最初に呼び出される関数となる。

【 0 0 4 0 】

(d) の部分は、ポインタ変数 f_p が関数 f を指示するように代入を行なっている。この代入は、図 3 で斜線の部分 3 0 1 が関数 f に対応し、符号 3 0 2 を付けた部分が変数 f_p に対応するメモリとすると、符号 3 0 2 を付けた部分に、斜線の部分 3 0 1 の最初のアドレスを格納することに相当する。

【 0 0 4 1 】

(実施形態 2：ポインタ保持部：ポインタ保持部は複数のポインタを保持してもよい)

なお、ポインタ保持部 1 0 1 は、複数のポインタを保持してもよい。

【 0 0 4 2 】

図 6 は、ポインタ保持部 1 0 1 が複数のポインタ 6 0 1、6 0 2、…、6 0 3 を保持している様子を例示している。ポインタが、それぞれが実行部 6 0 4、6 0 5、…、6 0 6 を指示している。また、図 6 に示すように、ポインタにより指示されない実行部 6 0 7、…、があってもよい。このような複数のポインタは、プログラミング言語上では、複数の変数によって表されたり、配列により表わされたりすることになる。

【 0 0 4 3 】

(実施形態 2：ポインタ保持部：ポインタはリスト構造によって保持されていてもよい)

また、ポインタ保持部が保持するポインタは、リスト構造によって保持されていてもよい。

【 0 0 4 4 】

図 7 は、ポインタがリスト構造によって保持されている様子を例示している。リスト構造の先頭のリストセル 7 0 2 を指すポインタ 7 0 1 があり、このポインタ 7 0 1 により指されているリストセル 7 0 2 から他のリストセルへのポインタをたどることにより、全てのリストセルがたどることができるようになっている。

【 0 0 4 5 】

特に、リスト構造の最後のリストセル 7 0 4 には、次のリストセルが存在しないので、リストセルへのポインタには、次のリストセルが存在しないことを示す特別な値が格納されている。この特別な値として、プログラミング言語では、例えば、N U L L という値が用いられる。

【 0 0 4 6 】

図 8 は、図 7 に例示したリスト構造を作るためのプログラミング言語上でのリストセルのデータ構造の定義を例示している。(a)の部分で定義される f p が実行部を指示するポインタであり、(b)の部分で定義される n e x t が次のリストセルへのポインタである。

【 0 0 4 7 】

(実施形態 1：ポインタ管理部)

「ポインタ管理部」102は、ポインタ保持部101に保持されるポインタを所定の変更条件に従って変更する。「所定の変更条件に従って変更する」とはあらかじめ決められた条件が成立した場合に変更するということであり、例えば、計算装置100に備えられた特定のボタンが押された場合、キーボードの特定のキーが押された場合、あるいは、キーボードから特定の文字列が入力された場合、計算装置100にデータの受信部が備えられており、その受信部によりデータが受信された場合、などに、ポインタを変更する。

【0048】

なお、「ポインタを変更する」とは、既に存在するポインタが別の実行部を指示するようにすることである。

【0049】

図5は、ポインタの変更を例示する。(a)において、ポインタ保持部101に保持されるポインタ201が実行部501を指示している。あらかじめ決められた条件が成立すると、(b)に例示されるように、ポインタ201が、実行部502を指示する。

【0050】

(実施形態 1：ポインタ管理部：プログラミング言語での表現)

プログラミング言語上では、ポインタ管理部は、あらかじめ決められた条件が成立したかどうかを判断し、もし、成立したならば、ポインタ変数に値を代入する記述により表現される。

【0051】

(実施形態 1：データ保持部)

「データ保持部」103は、前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能な部である。「前記ポインタの変更」とは、ポインタ管理部102によるポインタの変更を意味する。「前記実行」とは、ポインタ保持部101に保持されているポインタで指示されている実行部による計算の実行を意味する。「そのまま保持可能」とは、前記ポインタの変更によっても、データ保持部103で保持されているデータは変更されないことを意味する。「

変更」には、データそのものの変更のみならず、データの消去など、データが使用不能になることも含む。

【0 0 5 2】

したがって、データ保持部 1 0 3 は、データを保持するが、そのデータは、ポインタ保持部で保持されているポインタとは異なる領域で保持されていることになる。また、計算装置 1 0 0 がプログラムを用いて計算機によって実現される場合には、ポインタが変更されても、そのプログラムの実行が停止するようなことは無いことを意味する。

【0 0 5 3】

(実施形態 1：実行司令部)

「実行司令部」1 0 4 は、所定の実行条件に従って、データ保持部 1 0 3 で保持されているデータを利用して、ポインタ保持部 1 0 1 で保持されているポインタで指示されている実行部により計算の実行をさせるための部である。

【0 0 5 4】

(実施形態 1：実行司令部：所定の実行条件)

「所定の実行条件に従って」とは、あらかじめ決められた条件が成立した場合、ということを意味する。例えば、○特定のスイッチが押されたことが検出された場合、○キーボードから特定の文字列が入力された場合、○計算装置 1 0 0 が外部からデータを受信するようになっている状況において、特定のデータが受信された場合、○計算装置 1 0 0 が外部へデータを送信するようになっているとき、データが送信可能となった場合、○ある時点から一定の時間が経過した場合、などがある。なお、ポインタ管理部 1 0 2 の定義においても「所定の」という言葉が現われるが、実効司令部の定義における「所定の」と異なる意味であってもよい。

【0 0 5 5】

(実施形態 1：実行司令部：プログラミング言語での表現)

図 9 は、実行司令部 1 0 4 などをプログラミング言語で表現したものを例示している。(a)において、データ保持部 1 0 3 で保持されるデータを指示するためのデータへのポインタ変数 p が定義されており、(b)において、そのデータ

へのポインタ変数 p に $0x37468ABB$ という値が代入されている。(c)において、所定の実行条件が成立したので、ポインタ $f p$ の指示している関数を呼ぶ。その際、データへのポインタ変数 p の値を引数として与えている。これにより、ポインタ $f p$ の指示している関数が呼ばれ、データ保持部 103 で保持され、 $0x37468ABB$ で特定されるデータを用いて、計算が行なわれる。なお、図 9 では、一つのデータしかポインタ $f p$ の指す関数に渡されていないが、二つ以上のデータを渡すことも可能である。

【0056】

(実施形態 1：計算装置を実現するプログラム)

以上において、計算装置 100 の説明と並行して、計算装置 100 を実現するためのプログラムである計算プログラムについて、プログラミング言語上での表現を用いて説明を行なったが、ここで、再度、本発明における計算プログラムについて説明する。

【0057】

本発明における計算プログラムは、ポインタ保持ステップと、ポインタ管理ステップと、データ保持ステップと、実行命令ステップと、を計算機に実行させるためのプログラムである。

【0058】

「ポインタ保持ステップ」とは、計算の実行のための実行部を指示するためのポインタを保持するステップである。例えば、プログラミング言語において、図 4 のように変数 $f p$ が定義されている場合において、プログラムが起動したときに、ポインタに対応する変数 $f p$ の領域を確保するステップである。また、ポインタが図 7 に例示されるように、リストセルによって保持されている場合には、そのリストセルのメモリ領域を確保するステップである。

【0059】

「ポインタ管理ステップ」は、ポインタ保持ステップにより保持されたポインタを所定の変更条件に従って変更するステップである。例えば、所定の変更条件が成立した場合に、ポインタに対応する変数に代入を行なうステップである。

【0060】

「データ保持ステップ」は、前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能に保持するステップである。「前記ポインタの変更」とは、ポインタ管理ステップによるポインタの変更である。また、「前記実行」とは、ポインタ保持ステップにより保持されたポインタにより指示された実行部による計算の実行を意味する。「そのまま保持可能」とは、ポインタ管理ステップによりポインタが変更されても、データ保持ステップで保持されたデータは変更されないことを意味する。すなわち、ポインタ保持ステップで保持されるポインタが格納される領域と、データ保持ステップで保持されるデータが格納される領域とは、分離されている。

【0061】

「実行命令ステップ」は、所定の実行条件に従ってデータ保持ステップで保持されたデータを利用して、ポインタ保持ステップにより保持されたポインタで指示される実行部により計算の実行をさせるためのステップである。

【0062】

(実施形態1: 計算装置、計算プログラムで用いられる方法)

また、本発明においては、計算装置、計算プログラムなどで用いられる計算方法も提供される。

【0063】

その計算方法は、ポインタ保持ステップと、ポインタ管理ステップと、データ保持ステップと、実行命令ステップと、を含む。

【0064】

「ポインタ保持ステップ」は、計算の実行のための実行部を指示するためのポインタを保持するステップである。例えば、ポインタ保持部101を実行するためのステップである。

【0065】

「ポインタ管理ステップ」は、ポインタ保持ステップにより保持されたポインタを所定の変更条件に従って変更するステップである。例えば、ポインタ管理部102を実行するためのステップである。

【0066】

「データ保持ステップ」は、前記ポインタの変更によっても前記実行に利用されるデータをそのまま保持可能に保持するステップである。例えば、データ保持部 1 0 3 を実行するステップである。ここに「前記ポインタの変更」とは、ポインタ管理ステップによるポインタの変更である。また、「前記実行」とは、ポインタ保持ステップにより保持されたポインタにより指示された実行部による計算の実行を意味する。「そのまま保持可能」とは、ポインタ管理ステップによりポインタが変更されても、データ保持ステップで保持されたデータは変更されないことを意味する。

【 0 0 6 7 】

「実行命令ステップ」は、所定の実行条件に従ってデータ保持ステップで保持されたデータを利用して、ポインタ保持ステップにより保持されたポインタで指示される実行部により計算の実行をさせるためのステップである。例えば、実行命令部 1 0 4 を実行するステップである。

【 0 0 6 8 】

(実施形態 1：主な効果)

実施形態 1 において説明した計算装置、計算プログラム、計算方法によれば、所定の実行条件に従って実行される実行部が、所定の変更条件に従って変更することができるので、動的に実行部を変更することが可能となる。

【 0 0 6 9 】

(実施形態 2 (主に請求項 2 に関連する))

本発明の実施形態 2 においては、所定の条件に従って、ポインタの追加と削除を行なう計算装置、計算プログラム、計算方法が提供される。

【 0 0 7 0 】

(実施形態 2：構成)

計算装置について説明を行なうと、本実施形態における計算装置は、実施形態 1 の計算装置 1 0 0 のポインタ管理部 1 0 2 は、所定の条件に従ってポインタの追加と削除とを行なう機能を有する。

【 0 0 7 1 】

同様に、本実施形態における計算プログラムは、実施形態 1 の計算プログラム

ポインタ管理ステップにおいて、所定の条件に従ってポインタの追加と削除とを計算機に実行させるためのプログラムである。

【0072】

(実施形態2：ポインタの追加の例)

図10は、ポインタの追加の例を例示する。図10の(a)は、ポインタ保持部101が保持するポインタが、図7に例示されるようにリスト構造で保持されている様子を示している。このようにポインタがリスト構造で保持されている場合には、(b)に例示するように、新たなリストセル1009を確保し、そのポインタ1010が実行部1008を指示するようにして、リスト構造を指すポインタ1001がリストセル1009を指すようにし、リストセル1009からリストセル1002を指すようにする。

【0073】

図11は、図10の(a)の状態から図10の(b)の状態を得るための操作をプログラミング言語によって表わした例である。

【0074】

(a)において、新たに追加するリストセルを指す変数p1を宣言している。
(b)において、新たに追加するリストセルのメモリ領域を確保するために、mallocという関数を呼び、その結果を変数p1に代入している。なお、sizeof *p1は、リストセルがメモリ空間で占めるバイトを表わす。(c)において、確保されたリストセルのポインタが関数gを指示するような代入を行ない、確保されたリストセルから現在のリスト構造の先頭のリストセルを指すようにしている。pointerlistbaseは、変数であり、リスト構造を指すポインタ1001に対応する。pointerlistbaseに変数p1の値を代入することにより、図10の(b)の状態が得られる。

【0075】

なお、図10の(b)及び図11では、新たに追加するリストセルがリスト構造の先頭になるようにしたが、新たに追加するリストセルをリスト構造の任意の場所に追加することは容易である。

【0076】

(実施形態 2 : ポインタの削除の例)

また、ポインタがリスト構造に保持されている場合には、ポインタを削除することも可能である。

【0077】

図12は、ポインタを削除する操作をプログラミング言語により表わした例である。(a)において変数 `p1` を宣言している。(b)において、ポインタ `1001` に相当する `pointerlistbase` の値を `p1` に代入し、`p1` がリスト構造の最初のリストセルを指すようにし、`pointerlistbase` の値を、リスト構造の最初のリストセルの次のリストセルを指すようにしている。(c)において、`p1` が指しているリストセルの占めるメモリ領域を、`free` という関数を呼ぶことにより、破棄している。

【0078】

(実施形態 2 : 命令実行部の表現)

図13は、このようなリスト構造を用いた場合の命令実行部のプログラミング言語による表現を例示している。例えば、リスト構造の最初から3番目のリストセルのポインタにより指示される実行部を実行する場合には、(c)において、`pointerlistbase` の値を変数 `p1` に代入し、`p1` がリスト構造の最初のリストセルを指すようにする。(d)において、`p1` をリスト構造に沿って3回動かす。なお、`for (n=0 ; n<3 ; n++)` は繰り返しの処理を表わし、`p1=p1->next` を繰り返し実行することを表わす。`n=0` は、繰り返しの処理の前に `n` に0を代入することを表わし、`n<3` は、繰り返しを行なうために成立する条件を表わし、`n++` は、繰り返しを一度行なうと、`n` に代入されている値を1だけ増加させることを意味する。(e)において、変数 `p1` の指すリストセルに格納されているポインタにより指示されている関数を呼ぶ。その際、変数 `p` に代入されている値を渡す。

【0079】

(実施形態 2 : データ構造は、リスト構造に限られることはない)

なお、上記では、データ構造としてリスト構造を用いたが、リスト構造に限定されることは無い。例えば、充分大きな配列を容易してポインタ保持部を実現し

でもよい。また、ハッシュ構造や木構造を用いてポインタ保持部を実現するようにしてもよい。

【0080】

(実施形態2：主な効果)

ポインタ管理部によりポインタの追加が行なわれると、実効命令部により実行される実行部が増えることになるので、計算装置、計算プログラムにより提供される機能が増えることになる。また、ポインタ管理部によりポインタの削除が行なわれると、実行命令部により実行される実行部が減ることになるので、計算装置、計算プログラムにより、例えば不要な機能が提供されないようにすることができる。

【0081】

(実施形態3（主に請求項3、4に関連する）)

本発明の実施形態3においては、実行部の再読込がされる計算装置、計算プログラム、計算方法が提供される。

【0082】

(実施形態3：構成)

図16は、本実施形態における計算装置の機能ブロック図を例示する。図16は、実施形態1の計算装置100のポインタ管理部1002が実行部再読込手段1601を有している。

【0083】

(実施形態3：実行部再読込手段)

「実行部再読込手段」1601は、実行部の再読込を行なう。実行部の再読込は、計算装置100の動作開始時であってもよい。また、実行部の再読込は、計算装置100が動作している途中であってもよい。

【0084】

計算装置100が計算プログラムにより計算機により実現される場合には、実行部再読込手段1601に相当する機能は、例えばヒープ領域からメモリ領域を確保し、その確保されたメモリ領域に、関数や手続きを実現するインストラクションを含むデータを読み込むことにより実現される。

【 0 0 8 5 】

なお、「再読込」とは、全く同じ実行部を読み込むことに限定されることはない。例えば、バグなどの修正を行ったり、処理の高速化が図られたりすることにより、現在の実行部とは異なる実行部が読み込まれてもよい。

【 0 0 8 6 】

(実施形態 3：ポインタ管理部)

本実施形態において、ポインタ管理部は、実行部再読込手段 1 6 0 1 により再読込された実行部を指示するように前記ポインタを変更する。「前記ポインタ」とは、ポインタ保持部 1 0 1 で保持されるポインタである。なお、ポインタ保持部 1 0 1 で保持されるポインタの全てが変更されてもよい。あるいは、ポインタ保持部 1 0 1 で保持される一部のポインタが変更されるようになっていてもよい。

【 0 0 8 7 】

計算装置 1 0 0 が計算プログラムにより計算機により実現される場合には、このようなポインタ管理部の機能は、インストラクションを含むデータが読み込まれたメモリ領域の関数や手続きを実現するインストラクションの最初のアドレスをポインタに代入することにより実現される。このために、データに含まれる関数名とアドレスとの対応関係を示すシンボルテーブルが読み込まれたりする。

【 0 0 8 8 】

(実施形態 3：指示されなくなった実行部は削除されてもよい)

なお、実行部再読込手段により実行部が読み込まれ、その読み込まれた実行部をポインタが指示すると、どのポインタからも指示されない実行部が生じる場合がある。そのような実行部は、削除されるようになっていてもよい。

【 0 0 8 9 】

計算装置 1 0 0 が計算プログラムにより計算機により実現される場合には、どのポインタからも指示されない実行部のメモリ領域が回収され、次に実行部が読み込まれる場合に再利用されるようになっていてもよい。

【 0 0 9 0 】

(実施形態 3：実行部再読込手段は、データ変換実行部を読み込んでもよい（主に請求項 4 に関連する）)

なお、実行部再読込手段は、データ変換実行部を読み込むようにしてもよい。ここに、「データ変換実行部」とは、データ保持部101に保持されているデータを変換するための部である。

【0091】

「データの変換」とは、データの形式を変更することである。例えば、今までハッシュによって表現されていたデータをB木（B-tree）によって表現されたデータに変更することである。単方向リストを用いて表現されていたデータを、双方向リストに変更することも例示することができる。また、C言語の構造体によってデータの形式が定義されている場合において、構造体のメンバの順序の入れ替え、削除や、追加を行なうことも含まれる。また、メンバの型の変換を行なうことなども含まれる（例えば、`chanr`型のメンバの値を`long`型に変換する。）。

【0092】

構造体のメンバの追加としては、以下のものがある。すなわち、現在の構造体が次の式のように定義されているとする。

【0093】

【式2】

```
struct oldstruct {  
    int ver;  
    struct sockaddr src;  
    struct sockaddr dst;  
};
```

この時、新しい構造体の定義が、次のように、`time`というメンバが追加されたものとする。

【0094】

【式3】

```
struct newstruct {  
    int ver;  
    struct sockaddr src;
```

```
    struct sockaddr dst;  
    time_t time;  
};
```

この場合、実行部再読込手段が読み込むデータ変換実行部は、例えば、new struct のデータのメモリ領域を確保し、old struct のデータのメンバ ver、src、dst の値をそのまま持つように、new struct の ver、src、dst に代入し（なお、ver をデータ構造が変換された回数を示すメンバとする場合には、ver の値を 1 増やすなどのことを行なう。）、メンバ time の値を適宜生成して持たせることを行なう。また、old struct のデータが存在していた場所に new struct のデータを配置する。あるいは、old struct を指すポインタ変数の値を、new struct を指すように変更する。その後、old struct のメモリ領域を破棄などする。

【0095】

なお、データ変換実行部によるデータの変換の目的としては、例えば、実行部再読込手段により再読込された実行部の扱えるデータ構造が、それまでの実行部の扱えるデータ構造と異なる場合に、それまでの実行部の扱えるデータ構造のデータを再読込された実行部が扱えるようにすることを挙げることができる。

【0096】

したがって、このような場合においては、データ変換実行部は、実行部による計算の実行が行なわれる前に、データを変換する必要がある。そのために、例えば、データ変換実行部が読み込まれたときにデータ変換実行部によりデータの変換が行なわれるようにする。あるいは、ポインタ管理部 102 により、ポインタの変更が行なわれるときに、データ変換実行部によりデータの変換が行なわれるようにしたりする。あるいは、再読込された実行部による計算の開始を検出するようにしておき、その実行部による計算の開始が実行されると、データ変換実行部によりデータの変換が行なわれるようにしてもよい。

【0097】

（実施形態 3：データはバージョン情報を含み、実行部はデータに含まれるバ

ージョン情報に応じた計算の実行を行なうようにしてもよい（主に請求項5に関連する））

なお、実行部の再読込に伴う上記のようなデータの変換を不要にすることもできる。このためには、例えば式2あるいは式3で示したデータ構造の最初のメンバ `ver` は、データのバージョンを示すようにする。すなわち、前記データは、バージョン情報を含んでいてもよい。なお、ここでいう「前記データ」とは、データ保持部101で保持されているデータを意味する。また、「バージョン情報」とは、データ構造を示す情報である。すなわち、バージョン情報により、データ構造が指定される。バージョン情報の例としては、整数値、固定小数点数値、浮動小数点数値、文字列などの等しいかどうかを判断することができるデータを用いることができる。また、C++言語のようにクラスに等しいかどうかを判断するメソッドを持たせることができる場合には、バージョン情報を表わすクラスを用いてもよい。上記の式2、式3においては、`ver` という整数型のメンバにバージョン情報が保持されるので、整数値により、バージョン情報が表されている。

【0098】

このようにデータがバージョン情報を含む場合、実行部再読込手段1601により再読込された実行部は、そのデータのバージョン情報に応じた計算の実行を行なうようにしてもよい。すなわち、実行部は、まずデータに含まれるバージョン情報を調べる。次に、バージョン情報に応じた計算を行なう。プログラミング言語上（特にC言語上）では、このような処理は、データを、バージョン情報を保持するメンバを含む特定の構造体にキャストを行ない、データに含まれるバージョン情報に相当するメンバの値を得る。次に、そのメンバの値を `if` 文あるいは `switch` 文などにおいて参照し、適切な計算を選択して行なうようにする。

【0099】

また、もし、再読込された実行部がデータを生成した場合、そのデータ構造を示すバージョン情報をデータに含ませ、データ保持部101に保持させるようにしてもよい。

【0100】

このようにすることにより、実行部の再読込に伴うデータの変換を行なう必要が無くなる。

【0101】

(実施形態3：主な効果)

本実施形態によれば、実行部が再読込されるので、計算装置の実行が行なわれている途中で、バグが修正された実行部や、高速化が図られた実行部が読み込まれるので、計算装置を停止させることなく、実行部の更新（バージョンアップ）が可能となる。

【0102】

(実施形態4（主に請求項6に関連する）)

本発明の実施形態4においては、実行部が追加される計算装置、計算プログラム、計算方法が提供される。

【0103】

(実施形態4：構成)

図17は、本実施形態における計算装置の機能ブロック図を例示する。本実施形態の計算装置は、実施形態2の計算装置のポインタ管理部が実行部追加手段1701を有している。

【0104】

(実施形態4：実行部追加手段)

「実行部追加手段」1701は、実行部を追加する。実行部の追加は、実施形態3における実行部の再読込のように、計算装置がプログラムにより計算機で実現される場合には、例えばヒープ領域からメモリ領域を確保し、その確保されたメモリ領域に、関数や手続きを実現するインストラクションを含むデータを読み込むことにより実現される。

【0105】

(実施形態4：ポインタ管理部)

本実施形態において、ポインタ管理部102は、実行部追加手段1701により追加された実行部を指示するポインタをポインタ保持部101に追加する。こ

のポインタの追加は、例えば、図 11 に示したプログラミング言語で表現された操作により実現することができる。

【0106】

(実施形態 4：処理の流れ)

図 18 は、本実施形態における計算装置 100 の本実施形態に特有の処理の流れを説明するフローチャートである。ステップ S1801 において、実行部追加手段 1701 により実行部を追加する。ステップ S1802 において、ポインタ管理部 102 により追加する実行部を指示するポインタを、ポインタ保持部 101 に、追加する。

【0107】

(実施形態 4：主な効果)

本実施形態によれば、実行部が追加されるので、計算装置、計算プログラムを止めることなく、新しい機能を持った実行部を追加することができ、その実行部による機能が提供されるようにすることができる。

【0108】

(実施形態 5（主に請求項 7 に関連する）)

本発明の実施形態 5 においては、実行部が削除される計算装置、計算プログラム、計算方法が提供される。

【0109】

(実施形態 5：構成)

図 19 は、本実施形態における計算装置の機能ブロック図を例示する。本実施形態の計算装置は、実施形態 2 の計算装置のポインタ管理部が実行部削除手段 1901 を有している。

【0110】

(実施形態 5：実行部削除手段)

「実行部削除手段」1901 は、実行部を削除する。実行部の削除とは、実行部が配置されているアドレス空間のメモリ領域を回収し、再利用可能とすることである。

【0111】

(実施形態5：ポインタ管理部)

本実施形態において、ポインタ管理部102は、実行部削除手段1901により削除された実行部を指示しているポインタの削除又は変更を行なう。例えば、削除される実行部のアドレスが、0x284729ECである場合には、図21に示すようにリスト構造をたどり、0x284729ECを指示しているポインタを有するリストセルを探し出し、そのリストセルを削除する。あるいは、そのリストセルのポインタを別の実行部を指示するようにする。あるいは、NULLという特別な値にする。

【0112】**(実施形態5：処理の流れ)**

図20は、本実施形態における計算装置100の本実施形態に特有の処理の流れを説明するフローチャートである。ステップS2001において、削除する実行部を指示するポインタを探す。ステップS2002において、ポインタを削除または変更する。ステップS2003において、実行部を削除する。

【0113】**(実施形態5：主な効果)**

本実施形態によれば、実行部が削除されるので、計算装置、計算プログラムを止めることなく、機能を削除することができる。

【0114】**(実施形態1～5：応用例：ルータ)**

以下に、実施形態1から5の計算装置、計算プログラム及び計算方法の応用例を示す。その一つの例として、ルータがある。

【0115】

ルータは、通常は複数のネットワークインターフェースを有している。あるネットワークインターフェースでパケットを受信すると、データとして保持している経路情報に従って、パケットを送信するネットワークインターフェースを選択して、パケットを送信する。

【0116】

また、ルータは他のルータから送られてきた経路情報を受信し、データとして

保持している経路情報の再計算を行なう。また、他のルータに対して、自己がデータとして保持している経路情報を他のルータへ送信することも行なう。さらに、ある長さの時間の経過を検出するためのタイマーにより、経路の再計算を行なうべきことや隣接するルータがダウンしていることを検出したり、経路情報以外の制御情報の受信あるいは送信を行なうべきことを検出したりする。

【0117】

ルータは、このようなパケットを受信したこと、タイマーによりある長さの時間が経過したことなどの事象の発生を、「イベント」として検出するようになっている。イベントが検出されると、イベントに対応付けられた関数が呼ばれる。この関数のことを「イベントハンドラ」と呼ぶ。

【0118】

また、ルータの設定等を行なうための指示はルータに接続されたキーボードなどから行なわれ、これらもイベントハンドラにより処理がされる。

【0119】

(実施形態1～5：応用例：ルータ：ポインタ保持部)

従って、イベントハンドラを実行部とし、それを指示することをポインタによって行ない、これらのポインタを格納することによりポインタ保持部が得られる。

【0120】

(実施形態1～5：応用例：ルータ：ポインタ管理部、データ保持部、実行命令部)

また、このポインタを所定の条件に従って変更する部としてポインタ管理部が得られ、保持している経路情報などを保持する部として、データ保持部が得られ、イベントを取得した場合を実行条件として、そのイベントに対応するイベントハンドラを呼ぶ部を実行命令部とすることができる。

【0121】

(実施形態1～5：応用例：ルータ：フローチャート)

図14は、ルータの動作を説明するフローチャートを例示する。

【0122】

ステップS1401において、イベントの発生を待つ。例えば、selectシステムコールを実行し、パケットが受信されたこと、パケットの送信が可能になったこと、一定時間が経過したこと、などが発生するまで待つ。

【0123】

ステップS1402において、イベントの種類を決定する。例えば、selectシステムコールの結果を調べて、パケットが受信されたかどうか、送信可能になったかどうか、一定時間が経過したかどうか、などを決定する。

【0124】

ステップS1403において、イベントの種類に基づいて、データを取得する。例えば、パケットが受信された場合には、経路情報を取得する。

【0125】

ステップS1404において、イベントの種類に基づいて実行部を指示するポインタを取得する。

【0126】

ステップS1405において、データを利用してポインタで指示される実行部により計算の実行を行なう。例えば、制御パケットが受信された場合には、制御パケットの受信というイベントに対応付けられたイベントハンドラを呼ぶ。

【0127】

ステップS1406において、ポインタの変更を行なうべきかどうかを判断する。ここにいう、ポインタの変更とは、実施形態2などのポインタの追加、削除を含んでいてもよい。もし、ポインタの変更をすると判断された場合には、ステップS1407に処理が移行し、ポインタを変更することを行なう。ポインタの変更をしないと判断された場合には、ステップS1407をスキップする。

【0128】

ステップS1408において、ルータの処理を終了するかどうかを判断し、もし、終了しないならば、ステップS1401へ戻る。

【0129】

(実施形態1～5：応用例：ルータ：イベントとイベントハンドラの対応付け)

なお、ステップ S1403 と S1404 とにおいて、イベントに対応付けられたデータ、ポインタを取得する必要があるが、これは、図 15 (a) に示すデータ構造 1501 を用意することにより、取得することができる。データ構造 1501 は、イベントタイプ 1502、ポインタ 1503、データ 1504 からなっており、イベントタイプ 1502 は、イベントの種類を示し、ポインタ 1503 は、イベントに対応付けられたイベントハンドラを指示するポインタであり、データ 1504 は、イベントハンドラに渡すべきデータである。例えば、このようなデータ構造 1501 を、イベントタイプをキーとするハッシュ構造に格納しておくことにより、イベントタイプから、対応するデータ構造を高速に得ることができる。

【0130】

(実施形態 1～5：応用例：ルータ：イベントハンドラにバグがあってもルータを停止することなく修正することができる、イベントハンドラの機能追加もルータを停止することなく行なうことができる)

例えば、あるイベントハンドラにバグがあると判明した場合には、バグの無いイベントハンドラをルータに再読込させ、バグがあるイベントハンドラを指示していたポインタを変更し、再読込をしたイベントハンドラを指示するようにする。これにより、ルータを停止させることなく、イベントハンドラをバグが除去されたものに更新することが可能となる。また、イベントハンドラに機能が追加された場合に、その機能が追加されたイベントハンドラを再読込させることにより、ルータを停止させることなく、機能の追加が可能となる。ルータを停止させる必要が無いので、変数の値、データベースの中身、オープンしているファイル、通信あるいはソケットの状態がそのまま保持されたまま、イベントハンドラの更新、追加が可能となる。

【0131】

イベントハンドラが再読込された場合、そのイベントハンドラと、再読込される前のイベントハンドラと、が扱うデータの形式が異なる場合には、イベントハンドラとともに、データを変換するためのデータ変換実行部が読み込まれてデータの変換が行なわれるようになっていてもよい。

【0132】

また、この場合、イベントハンドラが再読込させたことを再読込イベントというイベントとして定義しておき、再読込イベントに対応付けられたイベントハンドラとして、データ保持部に保持されたデータの構造を更新する関数を用いるようにしてもよい。これにより、イベントハンドラの更新に伴うデータの構造の更新を行なうことが可能となる。例えば、データ構造がC言語の構造体によって定義されているとし、イベントハンドラが扱うデータを表わす構造体のメンバの順序が入れ替わったり、構造体メンバが削除されたり追加されたとする。この場合には、古い定義の構造体のデータから、新しい構造体のデータへ変換することを、再読込イベントに対応付けられたイベントハンドラにより行なうようにしてもよい。

【0133】

また、新たな種類のイベントに対する処理が必要になった場合には、そのイベントに対応付けられるべきイベントハンドラを読み込ませることにより、ルータを停止させることなく、イベントハンドラを追加することができる。

【0134】

また、反対に、イベントハンドラが不要になれば、イベントハンドラを削除することも可能となる。

【0135】

(実施形態1～5：他の応用例)

上記では、ルータを本発明の応用例として取り上げたが、本発明の応用範囲は、ルータにのみ限定されるものではなく、イベントを取得し、それに応じてイベントハンドラを呼ぶ計算装置、計算プログラム、計算方法に应用することができる。

【0136】

例えば、グラフィカルユーザインターフェースを持つプログラムにおいては、マウスの移動、マウスボタンの押下、キーボードのキーの押下、今まで他のウィンドウにより隠されて見えなかったウィンドウの一部が見えるようになったこと、などがイベントとして定義され、それぞれのイベントを処理するためのイベント

ハンドラが提供されている。したがって、イベントハンドラを実行部とし、それをポインタにより指示することにより、プログラムを停止することなく、イベントハンドラの再読込、追加、削除が可能となり、バージョンアップを行なうことが可能である。

【0137】

また、ウェブサーバなどにおいても、クライアントからの要求をイベントとして定義し、そのイベントにイベントハンドラを対応付けることができるので、本発明により、ウェブサーバを停止させることなく、イベントハンドラの再読込、追加、削除が可能となり、バージョンアップが可能となる。

【0138】

また、ウェブサーバの背後でデータベース管理システムが動作している場合が近年多くなってきた。データベース管理システムも、データの検索要求、データの削除要求、データの挿入要求などをイベントとして定義しておき、そのイベントに対応したイベントハンドラを対応付けることができるので、本発明により、データベース管理システムを停止させることなく、バージョンアップを行なうことができる。

【0139】

(実施形態1～5：他の応用例：実行部はイベントハンドラに限られることはない)

なお、上記の説明においては、実行部とイベントハンドラを同一視して説明を行なったが、実行部がイベントハンドラに限られることはない。例えば、C言語でプログラムが記述される場合、特定の機能を提供する関数を実行部として対応付けておき、そのように実行部に対応付けられた関数を直接呼ぶのではなく、関数へのポインタを用いて呼ぶようにしてもよい。もし、そのような実行部に対応付けられた関数にバグが発見されたり、関数の機能を拡張したりする場合には、バグを除去した関数や機能が拡張された関数を実行部として再読込させ、再読込された実行部に対応する関数を指示するように関数へのポインタを書き換える。

【0140】

このように、特定の機能を提供する関数を実行部とし、そのような実行部をポ

インタにより指示することにより、実行部をイベントと対応づけることなく、プログラムを停止させることなく、実行部の交換、追加などが可能となる。

【0 1 4 1】

【発明の効果】

以上のように、本発明によれば、停止させることなく、実行部の変更が可能となり、計算装置、計算プログラムのバージョンアップなどが可能となる。

【図面の簡単な説明】

【図 1】

計算装置の機能ブロック図

【図 2】

ポインタ保持部と実行部の関係を示す一例図

【図 3】

アドレス空間とそこに配置された実行部とポインタの一例図

【図 4】

実行部とポインタとのプログラミング言語での表現の一例図

【図 5】

ポインタ管理部によるポインタの変更を示す一例図

【図 6】

ポインタが複数存在する場合の一例図

【図 7】

ポインタがリスト構造に保持されている場合の一例図

【図 8】

リスト構造を構成するリストセルのプログラミング言語による表現の一例図

【図 9】

実行命令部のプログラミング言語での表現の一例図

【図 1 0】

リスト構造においてポインタを追加する一例図

【図 1 1】

ポインタを追加する操作のプログラミング言語による表現の一例図

【図 1 2】

ポインタを削除する操作のプログラミング言語による表現の一例図

【図 1 3】

特定のポインタを探す操作のプログラミング言語による表現の一例図

【図 1 4】

本発明によりイベントを処理するためのフローチャート

【図 1 5】

イベントと、ポインタ及びデータと、を対応付けたデータ構造の一例図

【図 1 6】

計算装置の機能ブロック図

【図 1 7】

計算装置の機能ブロック図

【図 1 8】

実行部を追加する処理のフローチャート

【図 1 9】

計算装置の機能ブロック図

【図 2 0】

実行部を削除する処理のフローチャート

【図 2 1】

削除される実行部を探す操作のプログラミング言語による表現の一例図

【図 2 2】

ルータによって構成されるネットワークの一例図

【図 2 3】

アドレス空間の使用の態様の一例図

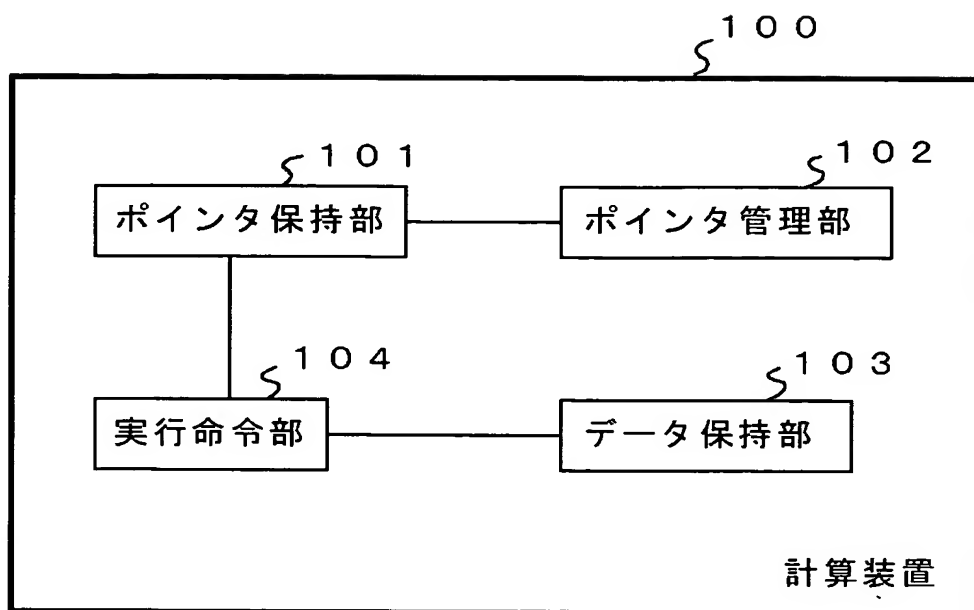
【符号の説明】

- 1 0 0 計算装置
- 1 0 1 ポインタ保持部
- 1 0 2 ポインタ管理部
- 1 0 3 データ保持部

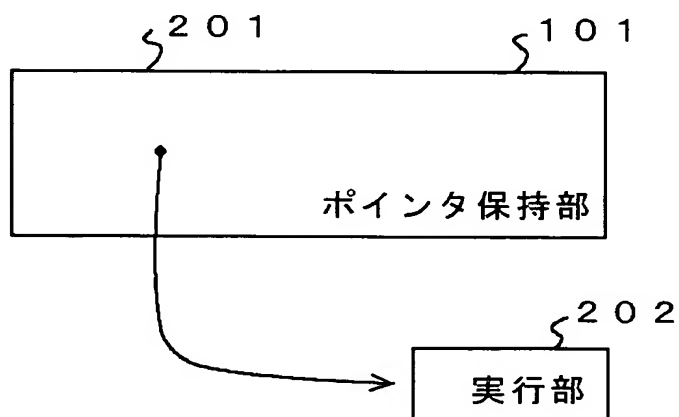
104 実行司令部

【書類名】 図面

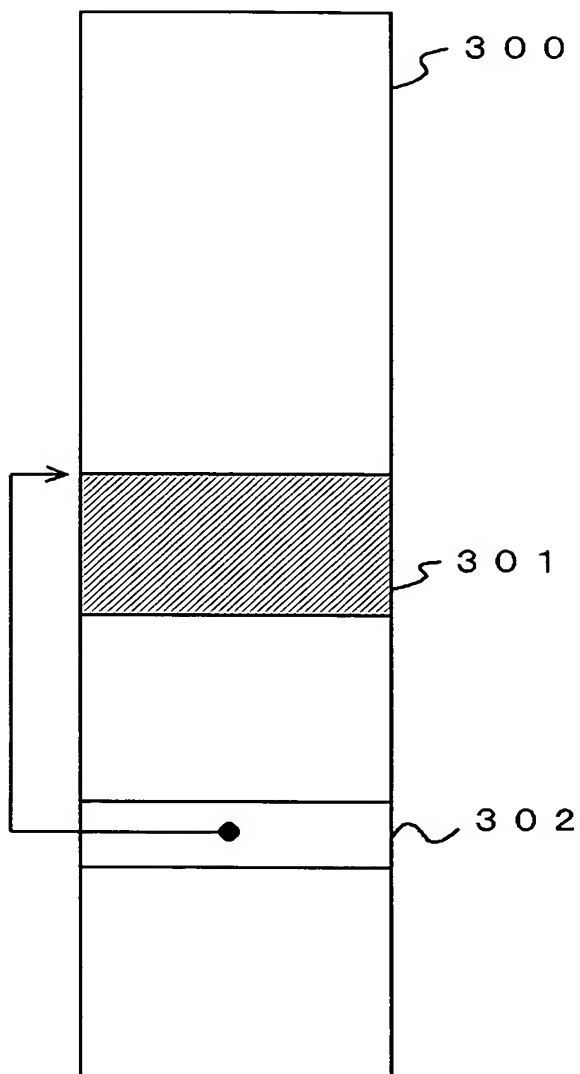
【図 1】



【図 2】



【図 3】



【図 4】

```
void (* fp) ();          . . .      (a)
```

```
void          . . .      (b)  
f(dp)  
void *dp;  
{  
    :  
}
```

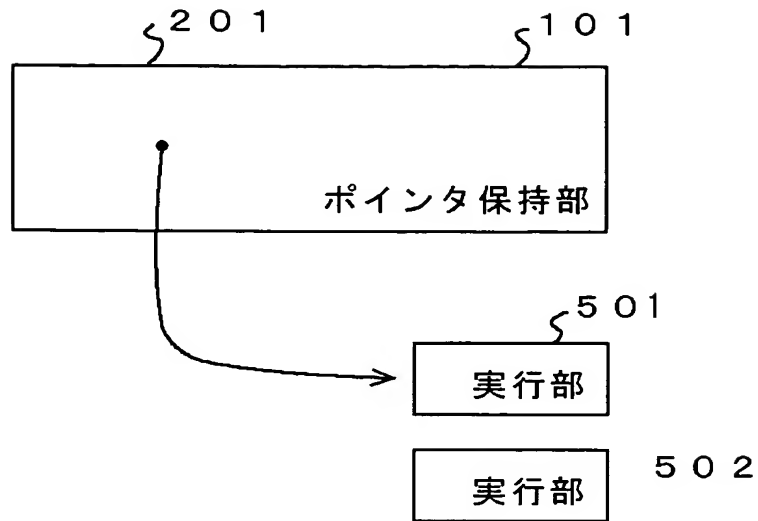
```
main()  
{          . . .      (c)
```

```
    :  
    fp = f;          . . .      (d)
```

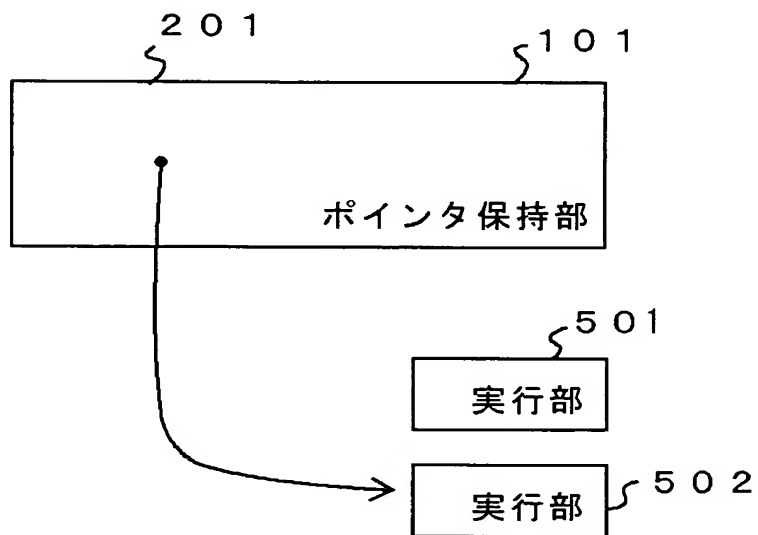
```
    :  
}
```

【図 5】

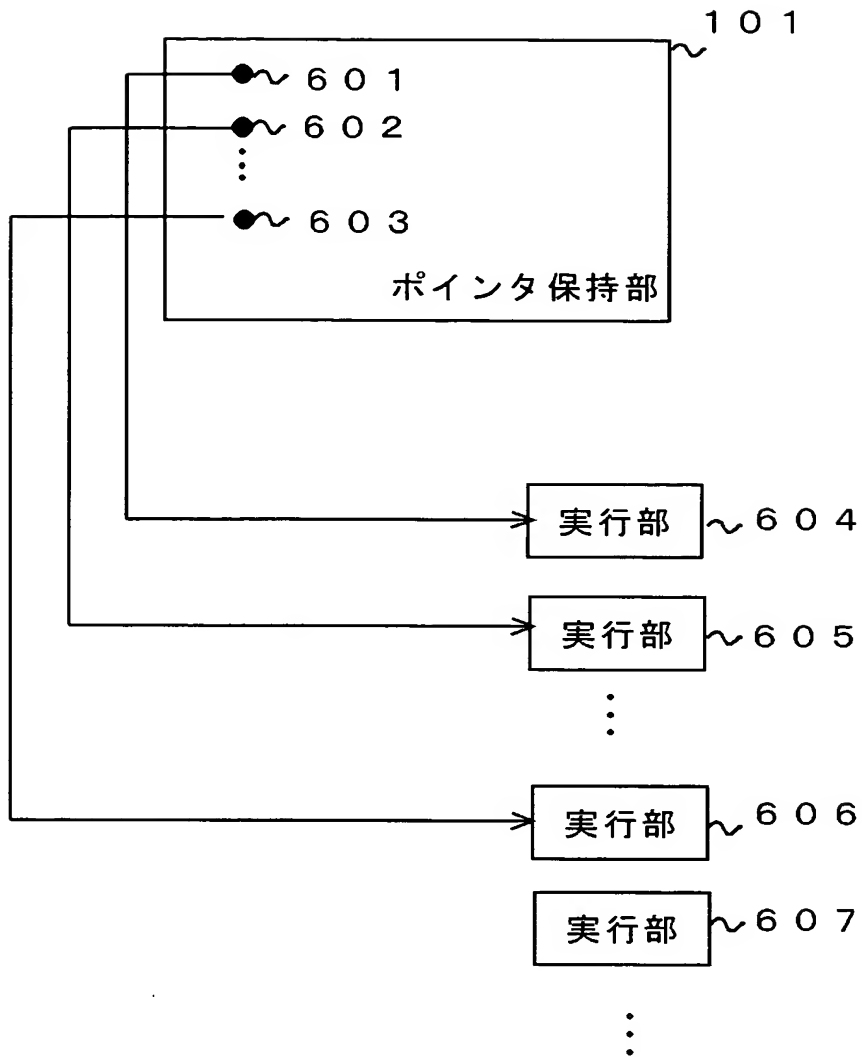
(a)



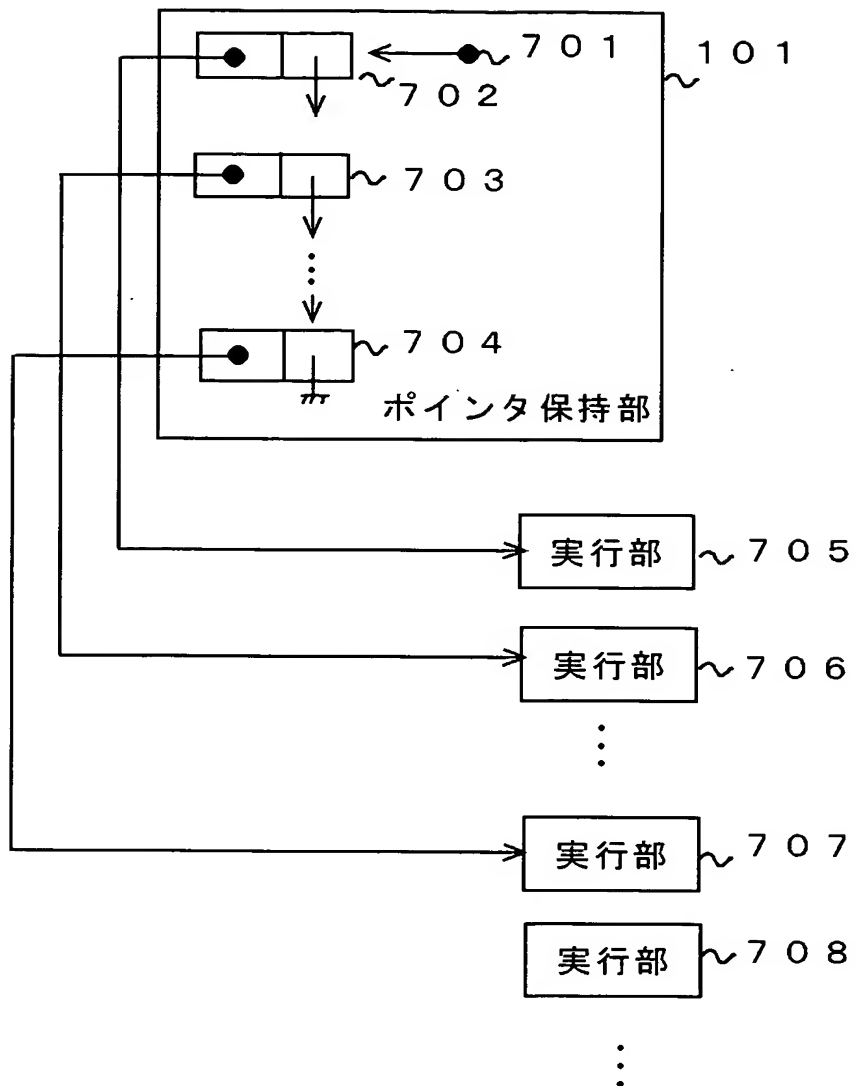
(b)



【図 6】



【図 7】



【図 8】

```

struct pointerlist {
    void (*fp)();          . . .      (a)
    struct pointerlist *next; . . .    (b)
} *pointerlistbase;

```

【図 9】

```
void *p ;           . . .           (a)
```

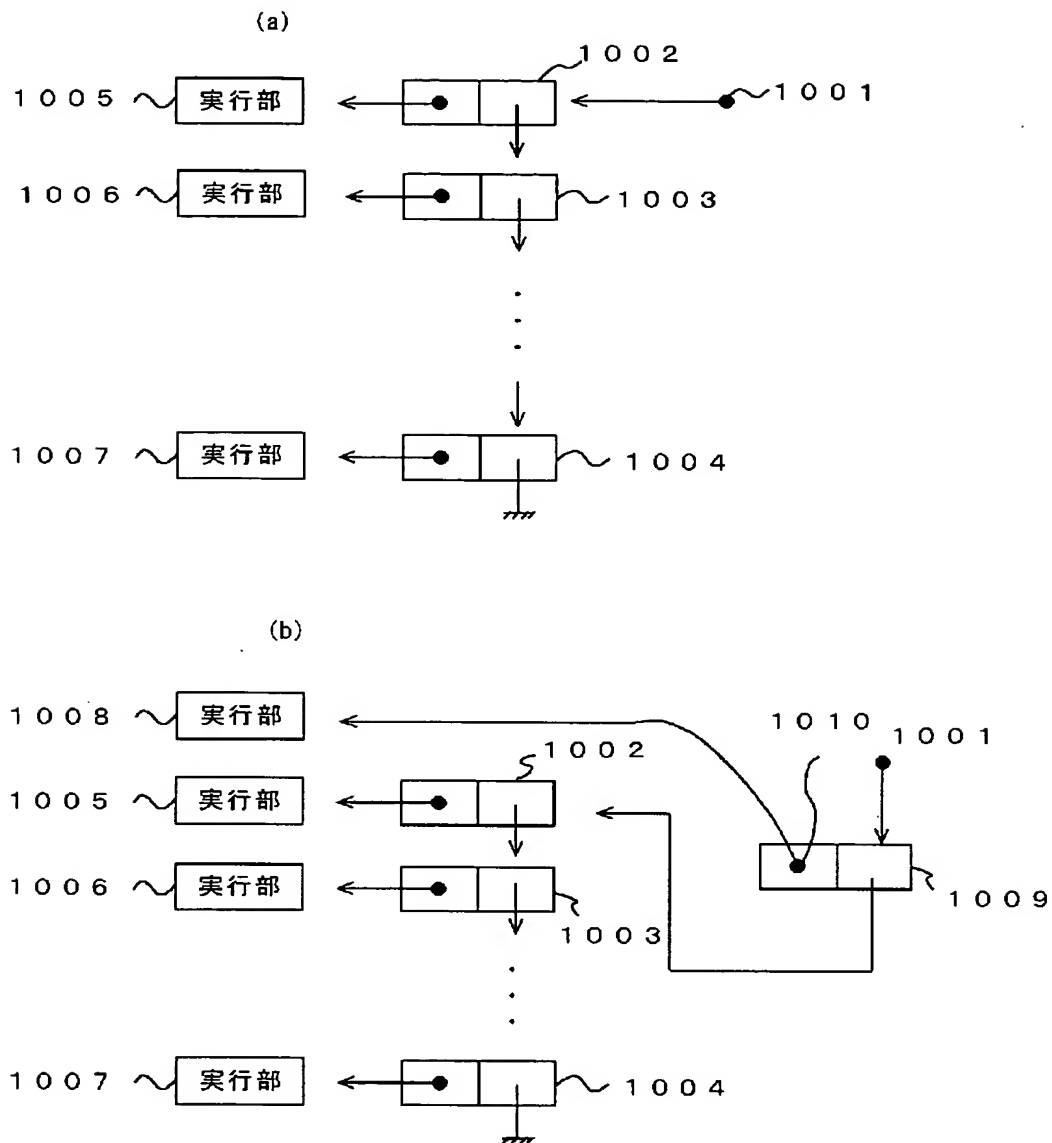
•

P = (void*) 0 x 37468AB8; . . . (b)

•
•
•

$$(*f_p)(p); \quad \cdot \quad \cdot \quad \cdot \quad (c)$$

【図 10】



【図 1 1】

struct pointerlist *pl; . . . (a)

pl = malloc (sizeof *pl); . . . (b)

pl —> fp = g;

pl —> next = pointerlistbase; } (c)

pointerlistbase = pl ;

【図 1 2】

struct pointerlist *pl; . . . (a)

pl = pointerlistbase; }

pointerlistbase = pointerlistbase —> next; } (b)

free (pl); . . . (c)

【図 1 3】

```
struct pointerlist *pl;
int                n;
void               *p;
} . . . (a)
```

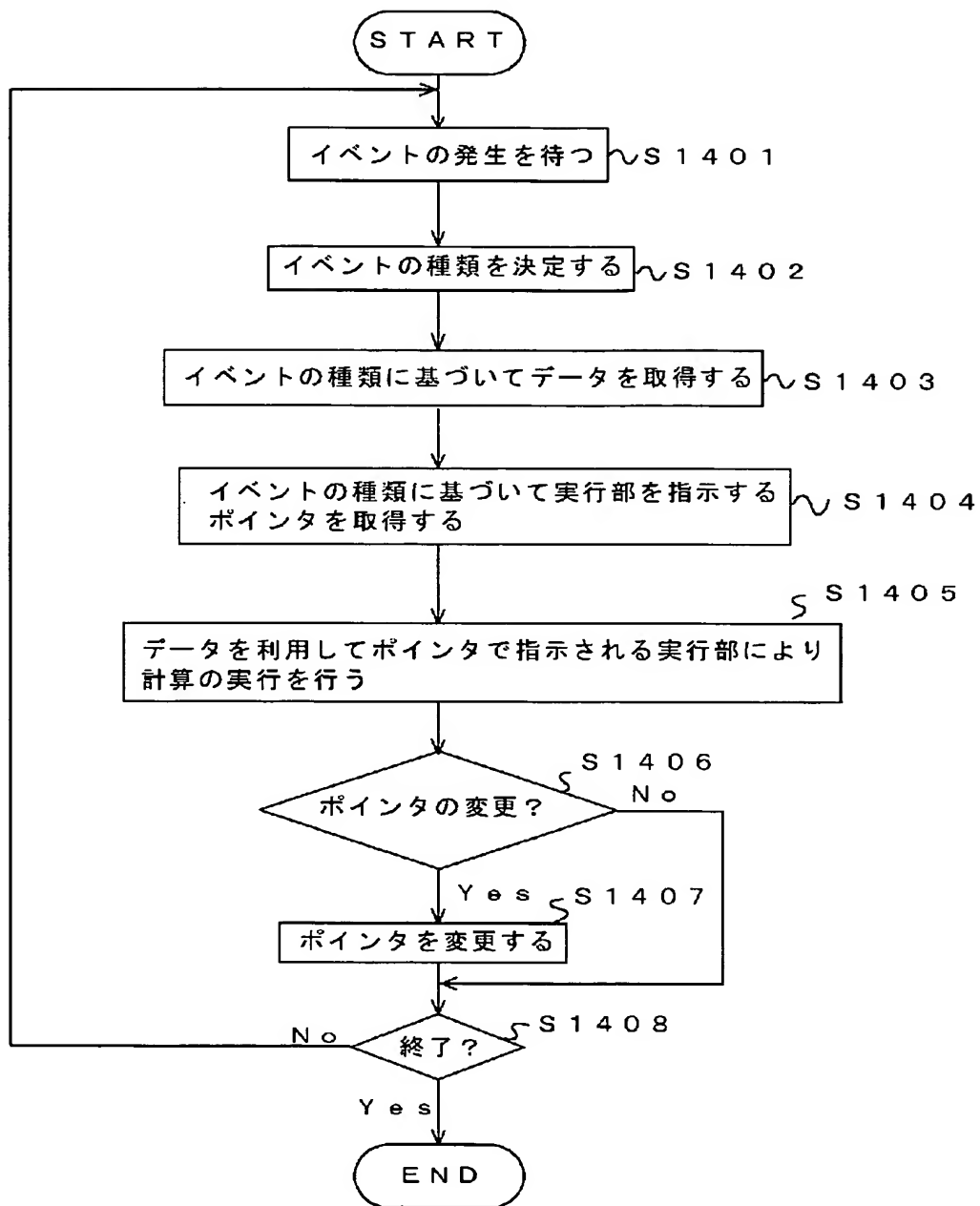
```
P = (void*) 0 x 37468AB8; . . . (b)
```

```
pl = pointerlistbase; . . . (c)
```

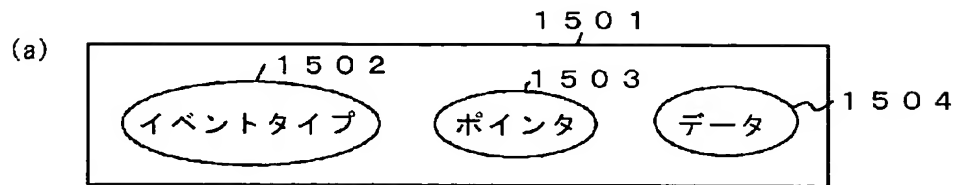
```
for ( n = 0; n < 3; n++)
    pl = pl -> next;
} . . (d)
```

```
( *pl -> fp) (p); . . . (e)
```

【図 14】



【図 15】

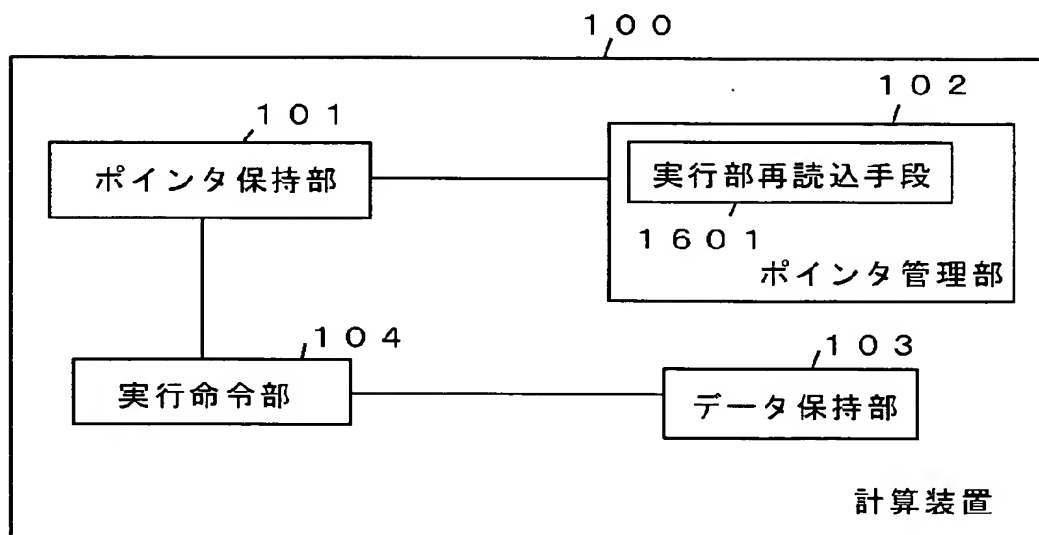


(b)

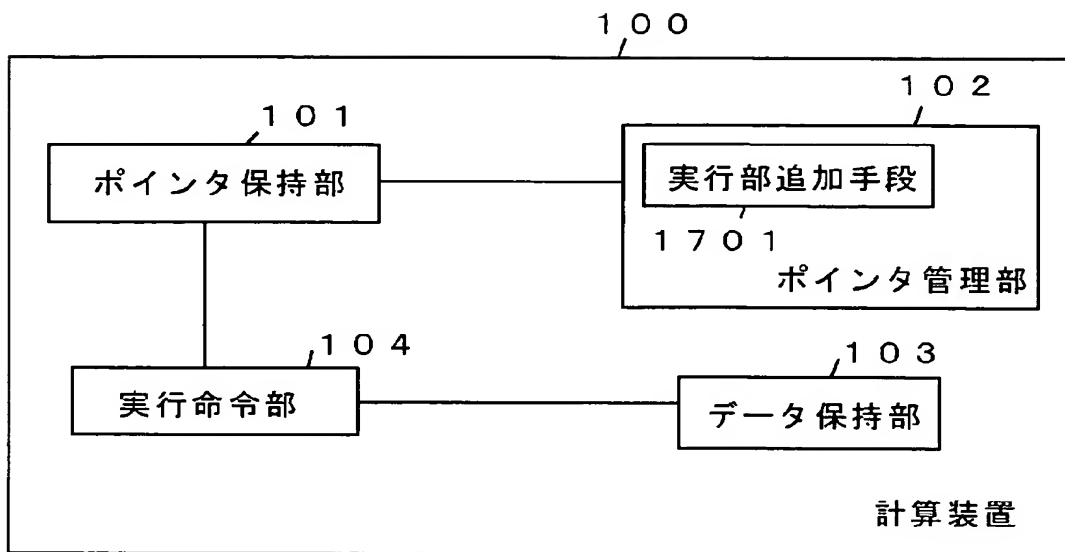
```

struct {
    enum eventType et;      . . . (イ)
    void (*fp) ();         . . . (ロ)
    void *dp;              . . . (ハ)
};
  
```

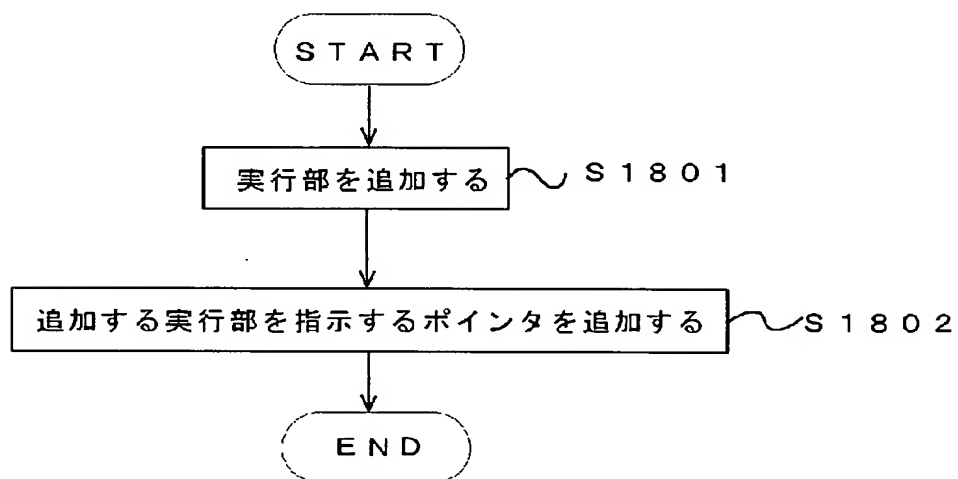
【図 16】



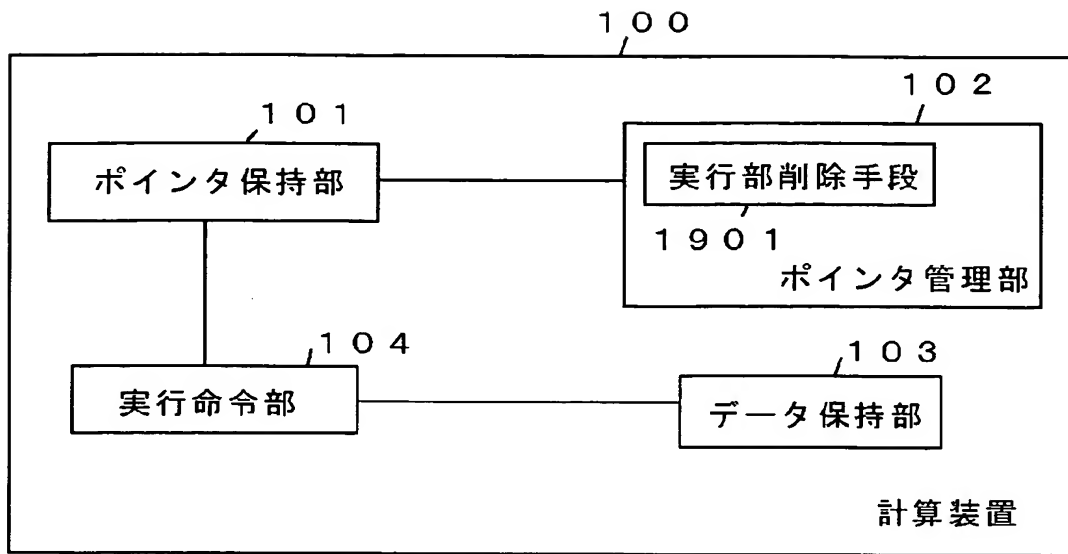
【図 17】



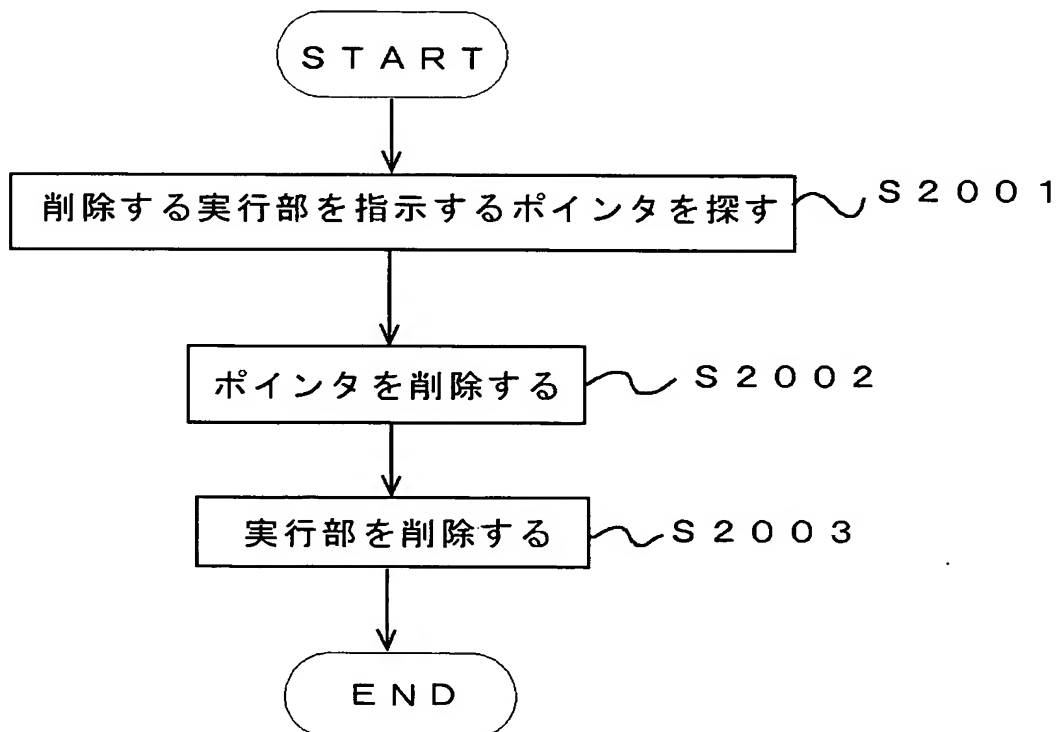
【図 18】



【図 19】



【図 20】



【図 2 1】

```

struct pointerlist *pl;
void (*ep)();
} . . . (a)

```

```

ep = (void (*)()) 0x284729EC; . . . (b)

```

```

pl = pointerlistbase; . . . (c)

```

```

while ( pl != NULL ) {
    if ( pl -> fp == ep )
        break;
} . . . (d)

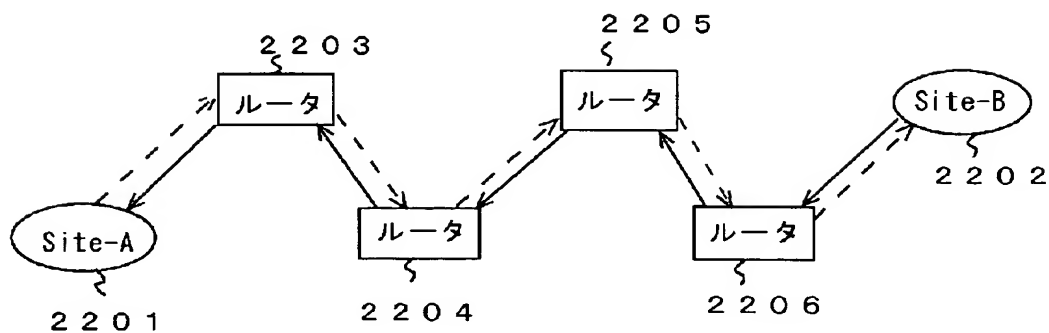
```

```

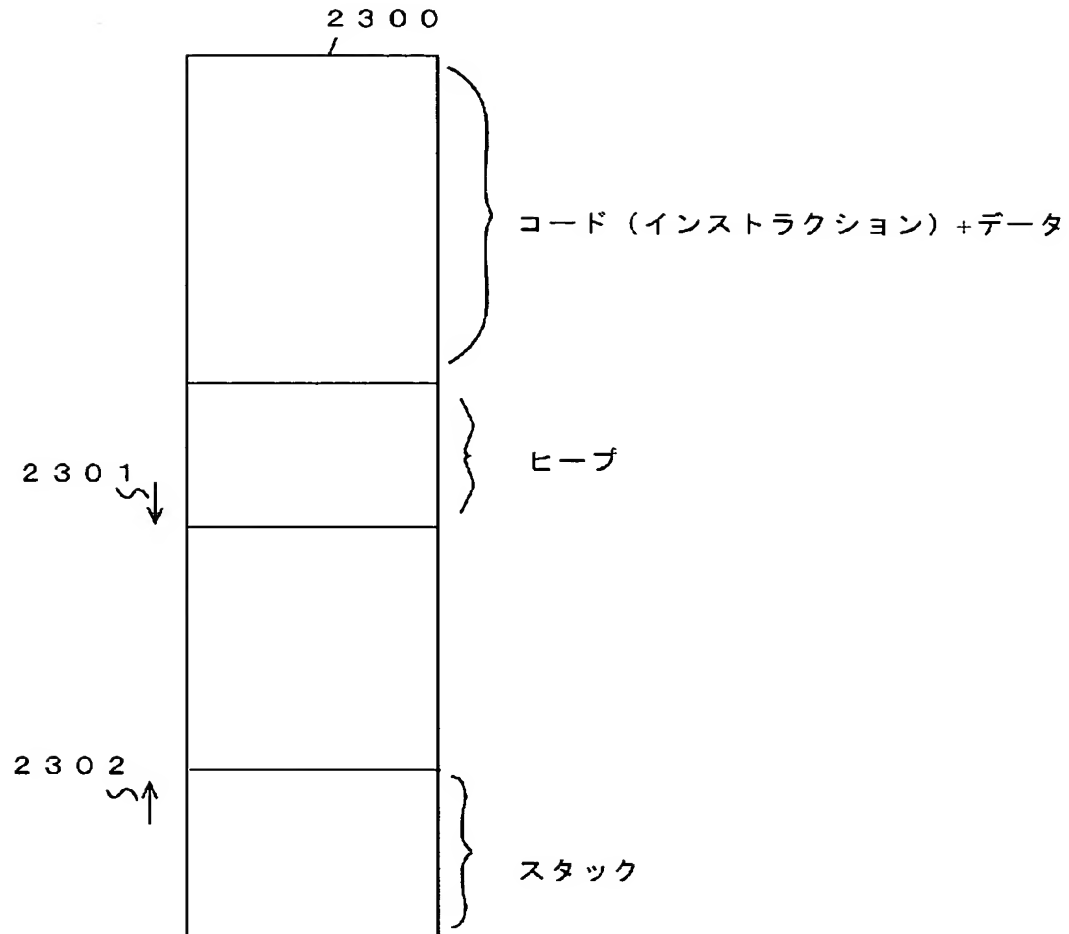
    pl = pl -> next; . . . (e)
}

```

【図 2 2】



【図 23】



【書類名】 要約書

【課題】 停止することなく、機能の変更・追加・削除ができる計算装置を提供する。

【解決手段】 計算を行なう実行部を特定するポインタを保持するポインタ保持部 1 0 1 と、ポインタ保持部で保持されているポインタを変更するポインタ管理部 1 0 2 と、実行部の実行に利用されるデータを保持するデータ保持部 1 0 3 と、データ保持部 1 0 3 で保持されているデータを利用して、ポインタ保持部 1 0 1 で保持されているポインタで指示される実行部に計算の実行をさせる実行命令部 1 0 4 を有する計算装置 1 0 0 を提供する。このような計算装置により、実行部の再読込、追加、削除に伴い、ポインタ管理部 1 0 2 によりポインタを変更することにより、動的に機能の変更・追加・削除が可能となる。

【選択図】 図 1

認定・付加情報

特許出願の番号	特願 2 0 0 2 - 3 6 6 3 3 7
受付番号	5 0 2 0 1 9 1 5 8 8 1
書類名	特許願
担当官	第七担当上席 0 0 9 6
作成日	平成 1 4 年 1 2 月 2 4 日

< 認定情報・付加情報 >

【提出日】	平成 14 年 12 月 18 日
-------	-------------------

次頁無